SuperGlue: Standardizing Glue Components for HPC Workflows

Jay Lofstead*, Alexis Champsaur†, Jai Dayal†, Matthew Wolf†, Greg Eisenhauer†
*Sandia National Laboratories †School of Computer Science, Georgia Institute of Technology

***Abstract*—The challenge for constructing HPC workflows is not the simulation, the analysis components, or the visualization tools. Usable, off the shelf tools to suit most needs are available. Instead, the labor and complexity is connecting these components together. Typically, an application scientist will write "glue" scripts that convert the output of one workflow phase to the input of the next. In nearly all cases, the output is written to disk after each phase, read and written for the "glue" conversion, and then read for the next phase. Even in online workflows, in which different components execute concurrently, custom glue code usually has to be written. This heavy lifting is one of the major stumbling blocks preventing standardized workflow frameworks from broad, general adoption. Attempts to streamline this process have not gained traction because of the necessity to still build these glue components. In essence, the workflow engine offered only a few, relatively small advantages over hand-coding the entire workflow.**

SuperGlue rethinks the custom glue components in the context of online workflows to offer *reusable glue components* that can bridge the type mismatches between the output of one workflow component and the input of the next. By performing a series of generic filter and selector operations in a typed environment, the same glue is usable, without modification, to connect components of different workflows with very different data types to achieve a plug-and-play workflow construction environment with known performance characteristics.

I. Introduction

As extreme computing architectures continue to evolve, it is increasingly clear that I/O is an increasingly significant problem. Projections suggest no significant parallel file system I/O bandwidth growth through the next 100x increase in compute capabilities. The traditional HPC approach of writing complete checkpoints or analysis dumps for later processing to gain scientific insights is becoming infeasible. Even with technologies like Burst Buffers, limited capacity reduces the usefulness. As such, in situ and in transit analysis and visualization toolkits with the capability to reduce, process, and otherwise mitigate the raw increase in data volume and throughput requirements are increasingly important.

Existing workflow engines, such as Kepler [10] and DAGMan [11], offer flexible ways to assemble components with rich functionality to manage the control flow. What they both lack is a way to easily deploy and manage the glue code required to connect the various components. One example illustrating the complexities comes from the Oak Ridge Leadership Computing Facility (OLCF). Kepler was used for several workflows for the fusion simulation users. While the initial goal was that an internal, expert resource would create the workflow, including glue components, it should be able to be maintained easily by the application scientists. Unfortunately, the expert was required regularly as the configuration evolved and scaled. The complexities of making and maintaining the glue components as the output shifted and managing the deployment was too high. Falling back to Python scripts managed by the application scientist proved easier and faster to maintain. While this approach using the parallel file system to stage intermediate data was sufficient, it is quickly becoming infeasible. The IO overhead for using the parallel file system is exceeding acceptable runtime percentages forcing a reduction in output and making scientific insights more difficult to discover.

To address the performance mismatch, Integrated Application Workflows (IAWs) are being developed. The easiest way to think of these IAWs is the Unix/Linux shell pipe operator to connect commands. The shell connects stdout of one program to stdin of the next with the assumption that each component in the chain can operate in this mode. For tasks at this scale, this approach works well. For the scientific workflows we are targeting, we have processes spread across potentially 10,000s of nodes connected to other components also running on multiple nodes. Unlike the command-line tools, none of these components generally have the ability to shift their input or output from how it is written to connect to a new online component without potentially significant code changes. The challenge with these workflows is dealing with the lack of a persistent store to stage intermediate data, interfaces for communicating data state and availability, and data organization changes required before a component can process data. Each of these has been investigated by different projects over the last several years.

Several frameworks have been developed that offer some functionality for supporting online workflows. The more advanced examples incorporate some data processing components, sometimes of limited scope, for performing in situ or in transit processing. Data Staging [15] and in particular data staging with processing capabilities [1], [16], are a solid step forward. The ADIOS IO library [9] was designed with this use case in mind. The HDF5 Virtual Object Layer (VOL) [2] was developed to support similar functionality.

Several efforts to work through some of the issues related to IAWs have been investigated [7], [25], [23], [4], [5], [27], and much on-going research in the space promises to expand on some of the needed techniques for management and placement. Some scientific codes have been addressing similar constraints for years by in-lining analytics functions and performing com-

plicated MPI communicator subdivisions to allow simulation and analytics to co-exist. One key observation, however, is that there is a lack of portability to the resulting implementations; they require a great deal of tuning and/or runtime placement control to make them function as desired.

This paper describes our work on SuperGlue, a set of generic, reusable components for composing scientific workflows. These are distributed data analysis and manipulation tools that can be chained together to form a variety of real-time workflows providing analytical results during the execution of the primary scientific code. Unlike existing components used in IAWs, SuperGlue components do not have a fixed data type. This one change enables using these components on completely different kinds of simulations that share nothing in their output format. Key to making this work is using a typed transport mechanism between different components. Many options exist for these transports and the particular mechanism selected is not critical.

While the main contribution of this work is the design of generic *glue code*, the *analytical* components we use in the workflows we present here are designed using the same approach. For example, an index selection tool is more of a glue component while a histogram component is more analytical. Still, we design both types of components as single MPI executables with similar input and output interfaces launched using similar configuration options. By blurring the line between generic glue components and generic analytical components, we present a flexible way to assemble scientific workflows without having to write any additional code.

The rest of the paper is organized as follows. First is a survey or related work in §II. Second, in §III, is a discussion of the two workflows we use to drive our insights. Next is an overview of the concepts behind SuperGlue in §IV. In §V, we provide an overview of the reusable components created. In §VI, we discuss the challenges of integrating reusable components. An evaluation is presented next in §VII. Finally is a discussion of Conclusions and Future Work in §VIII.

II. Related Work

Existing workflow systems have typically only been able to offer generic, reusable components when the workflow system is for a particular niche with a fixed datatype and standardized interfaces. For example, enterprise document processing systems may all work against a single database with each user only seeing their current worklist. As documents are processed, they are moved to the next work queue, or completed state, according to hand-coded rules [12]. The Workflow Management Coalition [24] has developed standards to make enterprise process workflows more portable. These standards are not intended to make components reusable, but to make different workflow engines able to inter-operate or to port a workflow from one engine to another. The actual communication interfaces and data types are left to the components themselves.

More directly related to this work and the scientific community are workflow engines and frameworks custom-made for the parallel computing environment. Pegasus [14] and DAGMan [11] work together to offer an engine to execute a workflow and a front-end system to construct the workflow process itself. This pair does nothing to address the actual connection between components. Instead, it is purely focused on providing a usable system for assembling components into a scientific workflow. This increment over a hand-crafted system should not be underestimated. It is a significant amount of work to make this work well.

Kepler [10] offers a nice GUI for assembling different kinds of scientific workflows. It offers different directors to manage how the workflow will be executed. Each component is an actor with channels connecting actors. Some simple decision channels are available to offer different execution paths given different output or return values from actors. As mentioned in the Introduction, complex workflows using Kepler have been assembled for many communities, including fusion science. Much like the Pegasus and DAGman pair, Kepler focuses on offering starting components and managing the control flow rather than offering standard interfaces and generic, reusable components.

To address much of the complexity of communicating between separate parallel components, in situ approaches are being investigated. Catalyst [7] offers a way to integrate the ParaView [13] analysis and visualization system directly into the simulation executable. Catalyst strips out many features to reduce the memory footprint and then requires explicit calls from the host application into Catalyst routines with predictable data types on in-memory data structures. While this can work for limited kinds of data processing, two limitations can cause problems. First, in an internal Sandia project in 2012, the CTH shock physics code used ParaView both in situ and also in transit. The in situ integration saw the executable grow from 30 MB to 300 MB and the scalability was strictly limited due to design flaws in ParaView for in situ use. While this project prompted the creation of Catalyst, this stripped down version of ParaView does not address all concerns. There is still a memory footprint overhead and a runtime pause in the simulation progress for the analysis and visualization to run.

Libsim [25] has a similar relationship to VisIt [20] as Catalyst has to ParaView. Both Libsim and Catalyst have a strict limitation that offline workflows do not. In both cases, because they are running on the same nodes as the simulation, time series analysis and visualization can be difficult or impossible. The potential scaling impact on the simulation because of limitations in Libsim or Catalyst can prevent their use for the most important use cases at extreme scale.

A middle path between in situ and offline processing was investigated in PreDatA [27]. This work demonstrates that the placement of the analytics can significantly affect the performance of workflows, and that this placement can be determined in part by the communication characteristics of the analytics components.

Glean [22], Nessie [17], and Mercury [21] are intended to facilitate offering portable workflows across different inter-

connect technologies. While their origins may not have been directly for addressing workflows, they have been re-purposed to address this field. Rather than offering anything related to managing data types, these tools simply offer a portable way to construct workflows.

A more active approach to managing workflow throughput was attempted in FlexPath [3]. Rather than focusing on the components themselves, FlexPath offers mechanisms to monitor input queues for workflow components and to redeploy components to reduce bottlenecks. It also has the ability to redirect output from an online workflow to disk in the case of an unrecoverable failure. We use ADIOS and FlexPath in this work.

Companion work to SuperGlue within this same project is Bredala [5]. This work presents an attempt to build a data model for in situ workflows. It has some similarity to FFS [6]. Unlike FFS, which is part of a much more complex infrastructure for typed messaging between distributed processes, Bredala is strictly focusing on the data model.

Overall, while all of these efforts are addressing different portions of the online workflows puzzle, none of them are addressing the idea of general, reusable glue and analytical components for scientific workflows.

III. Workflows

We designed and implemented two realistic real-time workflows based on scientific codes having large user bases: the LAMMPS Newtonian particle simulator [18] and GTCP, a particle-in-cell Tokamak simulator [8]. While both of these workflows eventually turn the simulation data into histograms of certain quantities of interest, how they arrive at their final result varies significantly. Creating similar types of results, and this using some of the same components but in significantly different ways, has allowed us to gain important insight into how best to design glue components that can be used in a wide variety of workflows. We first describe the workflows from a general point of view and then the components in greater detail, illustrated in Figure 3 and Figure 4.

A. LAMMPS Workflow

We configure LAMMPS to simulate a disruption (a "crack") in a thin layer of particles and output 5 numerical properties describing each particle in the simulation at regular timestep intervals. This corresponds to two-dimensional data (particles as one dimension and properties of interest as another) and among these properties are the three-dimensional components of the particles' velocities. In this case, the workflow outputs one histogram per timestep showing the velocity distribution for all simulation particles at that point in time. From the LAMMPS output, the particular quantities of interest must be extracted and then a histogram generated.

B. GTCP Workflow

GTCP, a code that simulates a toroidally confined plasma, splits the solid into toroidal slices, each made up of a number of grid points. For each of these grid points, it outputs 7 properties of the plasma such as pressure and energy flux. This division of the toroid is illustrated in Figure 1. The output of the simulation is therefore a three-dimensional array in which the dimensions span: (a) toroidal ranks (toroidal slice number), (b) grid point numbers, and (c) property indices. In this workflow, the per-timestep histogram generated shows the distribution of per-gridpoint parallel pressure across the entire simulation. From GTCP's output, the particular quantities of interest must be extracted and then a histogram generated.

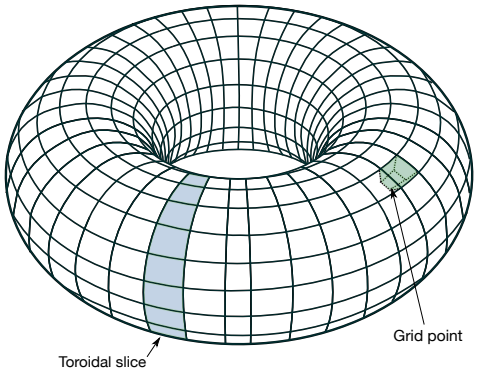

Fig. 1. Representation of GTCP Toroid, modification of [26]

C. Discussion

In both cases, a particular subset of interest is extracted from the output data set, a calculation is performed, and a histogram is generated. This is illustrated in Figure 2. In many cases, the histogram component may be some standard, reusable operator. The challenge is the data selection and mathematical manipulation to obtain the quantities of interest in a way that (a) presents an intuitive interface to the scientist constructing the workflow and (b) is useful in different types of workflows in which data have different sizes and dimensions. This difficulty arises because (a) the data at each stage of the workflow is distributed over the collection of processes involved in each component even if it forms a coherent whole and (b) the extraction and re-arrangement of multi-dimensional, distributed data, in a way that is configurable by the user at runtime is challenging. Both workflows are similar, insofar as both codes generate a large output while we are only interested in a per-timestep histogram of a particular simulation quantity. However, because the simulations generate very different data, selecting the relevant data from the simulations and formatting it in a such a way that the final Histogram component can operate on it is done very differently in each workflow.

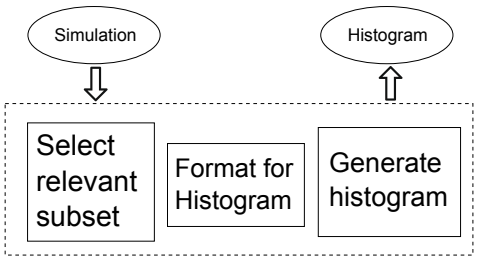

Fig. 2. Generic Workflow Illustration

In typical scientific workflows today, custom glue code

is written for selecting relevant data and writing it so the "formatter" illustrated in Figure 2 can work. Then, potentially additional custom glue code will fix the histogram calculation into something that can be rendered or saved as desired. In this work, we demonstrate general, reusable components capable of handling all three intermediate operations.

The workflows presented here use no custom glue code. Instead, the glue is designed as generic components that accept command-line configuration options at launch time. At most, the user specifies a few parameters and organizes the components into a proper pipeline. Both of these operations are easy enough that a non-expert can create workflows through GUIs or other guided assembly techniques.

That we are able to use some of the same glue code to connect the start to the end of each workflow shows that generic glue code that can manipulate large datasets in real time is possible and useful for assembling very different workflows. In addition, we later show that generic components, both glue and analytical, have the advantage of possessing known performance characteristics, which greatly facilitates the configuration of workflows that use them.

IV. Design

In this work, we offer some insight in the design of generic data manipulation and analysis components from our implementation of two workflows. These workflows are driven by two different scientific codes, yet they share some of the same components. First, we present our insights and then discuss our implementation choices and how they affect the project.

A. Insights: Overview

By evaluating the presented workflows and considering other workflows with which the authors are familiar, four particular insights are revealed.

1) To allow for the greatest variety of workflows, data manipulation primitives and data analysis components should be packaged in similar ways – that is, regardless of their individual complexity, the pieces that make up these workflows should export compatible interfaces as much as possible.

2) The ability to handle multi-dimensional data, along with the consistent labeling of dimensions and quantities as metadata, allows for components that are highly adaptable and simple to use.

3) While different types of components understand varying levels of semantics, maintaining a high level of semantics (i.e., labeling quantities and dimensions as much as possible) early on and when passing through components that do not necessarily require all of these labels allows for the most functionality downstream.

4) Because programming languages understand multi-dimensional data as being in a specific order in memory, there is a need for glue components that re-arrange data and re-label its dimensions without necessarily changing its size. Indeed, when data is stored in a database on disk, it is simple to gain a desired view of the data, for example by using SQL. However, in the middle of a real-time workflow, data must be presented to the components in a format that they expect and understand. This requires a specific ordering of data in memory. We expand on this topic in §IV-D.

These insights guide the design for the reusable glue and analysis components presented in this paper. From a general perspective, designing a smaller number of components to assemble workflows with finer step decomposition allows for more general processing and more accurate performance expectations than designing more numerous components each having more complex functionality.

B. Multi-Dimensional Data Support

Many scientific codes serialize their output, effectively packing multi-dimensional data into a single dimension. However, this technique offers little information on the data to downstream components in a workflow.

For example, both of the simulations used in the workflows presented in this work pack their output into one-dimensional arrays, even though the output LAMMPS is logically two-dimensional, and that of GTCP three-dimensional. In order to use the same glue code to extract desired data from both outputs, this code must be able to operate on two-dimensional data as well as three-dimensional data. In fact, if the glue code is able to operate on any number of dimensions, all that must be provided to it at runtime is the dimensions and indices to extract. This is precisely the role of our *Select* component.

In general, it is advantageous to (1) design components that can operate on multi-dimensional data as much as possible, and (2) format the output data of scientific applications as having well-defined dimensions. Emphasizing the support for multi-dimensional data in the design of workflow components allows for maximum compatibility between the interfaces of components by providing a consistent way to refer to the data.

Still, while multi-dimensional data support provides a consistent way to refer to the data, not all components should be designed so as to work with ***any number*** of dimensions. For example, we found it advantageous to design our *Histogram* component to work with only one dimension. Indeed, creating a histogram from one-dimensional data is intuitive. Supporting a higher number of dimensions would add unnecessary complexity to this component. If the data has more dimensions than are expected, and only particular indices in a particular dimension hold the data of interest, we can simply use filter and selector glue to extract and format the data for *Histogram* to operate on.

C. Semantics

When data is organized under clearly defined dimensions, labeling these dimensions and the indices they hold as the data goes through each component lets downstream subscribers refer to dimensions using their names. Because the data is potentially re-sized and re-arranged in the course of a workflow, it is useful to maintain such semantics as much as possible. However, the absence of labels should not block the workflow execution.

For example, in both workflows, we modified the simulations so that they label the quantities that they output in the dimension that holds these quantities. This lets the *Select* component extract the quantities of interest by referring to them by name. However, we did not label the dimensions themselves, rather referring to them by number.

When they exist, these labels are concatenated into a header string passed along to the next component in the workflow as another variable through the ADIOS interface, which allows for the arbitrary output of any number of variables of any type alongside the main simulation data. While in our current implementation, *Select* is hard-coded to always looks for such a header describing the quantities in the dimension of interest, we can easily extend this functionality to let the user either refer to dimensions and indices by name, when headers exist, or by number, when they do not.

In general, labels should be used as much as possible, but they should also be kept *optional*.

D. Dimension Reduction

As discussed earlier, there is a need in scientific workflows with fine step granularity for glue components that re-arrange and re-label data without necessarily changing its size. This is illustrated by the need for the *Dim-Reduce* operation in the GTCP workflow.

In its raw output, GTCP keeps track of the toroidal slice that produces the data of interest by using a dimension that spans the "Toroidal rank" of grid points. In our workflow, we wish to create histograms encompassing all grid points in the toroid, thereby eliminating the concept of "Toroidal rank" along the way and instead growing the dimension that spans the number of grid points.

Programming languages represent multi-dimensional arrays in a specific order in memory, so we cannot simply keep the data ordered as it is, change the number of dimensions and their sizes, and assume that the new dimensions correctly reference the data.

The dimensions of GTCP's raw output are $N_0 \times N_1 \times N_2$, where:

- N_0 is the size of the *toroidal rank* dimension D_0
- N_1 is the size of the *gridpoint* dimension D_1
- N_2 is the size of the *property* dimension D_2

At one stage of the GTCP workflow, we wish to eliminate the concept of *toroidal rank* of the data. This is to bring the data one step closer to a format that *Histogram* understands: one-dimensional data. For the data to be one-dimensional, two instances of the *Dim-Reduce* glue operation are required. We focus here on the first of the two.

The desired result of this operation is that:

- The concept of *toroidal rank* of a particle disappears
- The concept of *gridpoint number* loses its original meaning and takes on a new one; the indices in this dimension now cover all gridpoints in the toroid, rather than only those in a particular toroidal slice
- The concept of *property* keeps its original meaning, and the size of its dimension is unchanged

We can say that D_1, the *gridpoint* dimension, ***absorbs*** D_0, the *toroidal rank* dimension. The array resulting from this operation has dimensions $N_1' \times N_2$, where:

- $N_1' = N_0 \times N_1$ is the new *gridpoint* dimension D_1' size
- $N_2' = N_2$ is the *property* dimension D_2' size

We call this operation *dimension reduction*. Even though it does not change the size of the data set, we have seen in our implementation that it often changes the in memory data ordering. Consequently, it is still potentially a compute-intensive operation with sizable communication overhead when the dataset is large and many processes are involved in it.

This operation fits well into a SuperGlue component, and it is precisely the role of the *Dim-Reduce* component, which can operate on a dataset having any number of dimensions.

E. Implementation Artifacts

While particular implementation decisions are made to investigate the potential for reusable glue and analysis components, the choices made in the tools used to build the components and workflows presented in this work are by no means the only tools and techniques for achieving success. Rather, the selected technologies offer many features that facilitate the creation of reusable components. The reasoning for these selections is explored below.

Though we refer to the SuperGlue components as single entities, they are distributed codes able to split computation on large dataset among the processes of which they are composed. More precisely, they are MPI executables written in C where all processes in a component belong to the same MPI communicator.

Since we need some analogy to the Linux shell pipe operator for connecting components, we choose to use ADIOS [9] for the flexibility in writing destinations, reading sources, and offering a typed data stream. In particular, we use the FlexPath transport. It implements a *stream-based* data exchange abstracted to the components through the ADIOS interface, is asynchronous, and allows for data exchange between any number of writers and readers. Therefore:

1. We can launch components of the workflow in any order: downstream components will wait for the availability of data from upstream components and upstream components will buffer data up to a certain size until they are able to send it downstream. This also means that the decision as to which downstream components to use can be made after the upstream components have started running allowing for real-time adjustments to the workflow based on results obtained upstream.

2. Even if the number of processes used for one component is different from that used for the previous one in the workflow, each component can split the data (and therefore the computation) evenly among its processes. We should mention, however, due to the current implementation of FlexPath, there is overhead data exchanged when different numbers of writers and readers are used. Even if reader R requests only a portion of writer W's data, the current implementation is such that W sends all of its data to R. This is in the process of being

improved, and this effort is orthogonal to the work done for this project.

In addition, ADIOS and its transports, such as FlexPath, keep track of the data dimensions and their sizes. Therefore, when a component receives a multi-dimensional array, it can discover the dimensions of the data and their sizes as defined by the previous component in the workflow. Note that the data type of the input to one component may be changed for the output. This is crucial for operators that select a data subset or generate a derived product.

There is no need to re-compile SuperGlue components when using them in different workflows. Any configuration of a component for a particular workflow can be provided through run-time arguments. For all components, the user must specify the input stream name from which to read, the input stream array name containing the data on which to operate, the output stream name to which to write, and the array name to use in the output stream. Referring to streams and arrays using names allows users to easily chain together these components into potentially complex workflows, providing a similar advantage to that of labeling dimensions and indices, as previously discussed.

V. Reusable Components

This section provides greater details about the individual SuperGlue components and how a small number of parameters allow the them to operate (a) on a variety of different input data formats, and (b) in a user-specified way.

A. Select

Given an input stream that includes an array with any number of dimensions, Select extracts certain indices from one of the dimensions. Thus, it outputs an array with the same number of dimensions, but with the dimension of interest having a smaller size. In order to select the quantities of interest, the component uses a header which must be passed by the previous component in the workflow. The header is a list of strings that name the quantities in the dimension of interest. This allows for easy selection of quantities at runtime when Select is launched. For example, in the LAMMPS workflow, the simulation outputs the ID, Type, Vx, Vy, and Vz of each particle, where Vx, Vy, and Vz are the components of the three-dimensional particle velocity. Select discards the ID and Type, building a new array consisting only of the velocity components. The user, or a higher-level dataflow assembler, must pass to this component the index of the dimension from which to select, as well as the indices of the quantities of interest within that dimension.

B. Dim-Reduce

Dim-Reduce is a glue component that removes one dimension from its input array, "absorbing" it into another dimension without modifying the total size of the data. The other dimensions are left unchanged. This component can work with an input array having any number of dimensions. The output is an array with one dimension removed and with another dimension that has been re-defined. When using this component, the user must specify which dimension to eliminate and which to grow.

C. Magnitude

In our current implementation, Magnitude expects a two-dimensional array as input, where one dimension spans the data points at each time step (particles in the case of LAMMPS and grid points in the case of GTC) and the other dimension spans any number of components of the same vector, for example the three-dimensional components of velocity in the LAMMPS workflow. Magnitude calculates the magnitudes of the vectors from the values of their individual components and outputs a one-dimensional array of the new values. Which dimension is which in the input array is specified by the user at runtime. A small number of changes and a few start-up parameters could generalize this code to perform any number of common operations that calculate a quantity from many, applying a known formula over a two-dimensional dataset, thus allowing this component to fit into a variety of scientific workflows.

D. Histogram

The processes that make up the Histogram component partition among themselves a one-dimensional array of data. They communicate to discover the global minimum and maximum values in the array, create a number of bins between these two extremes, and then communicate again to count the number of values in the globally partitioned array that fall in each bin. The number of bins to use must be passed to the component when it is launched.

In our current implementation, one of the processes of Histogram writes the output to a file on disk. We chose this approach because this component is generally used as an endpoint in the workflow and because the output of this component is generally very small compared to its input and can be easily written by a single process. However, as we see now, letting this component output its data in the same way as the other components, that is as an ADIOS stream, and instead writing to disk when needed using a component specifically designed for this purpose would provide greater flexibility.

E. Dumper

While this component was not created in time for this paper, the value of a component specifically designed to be the endpoint of a workflow is clear. The key goal for this component is to offer a way to write an ADIOS stream into an output file using some particular format. Whether to write the workflow output as HDF5, ADIOS-BP, or a simple text file could simply be selected by the user as an option, requiring no modifications to existing components, and no re-compilation.

Another component that would be of value would be one with a graph plotting capability. For example, GNU Plot [19] takes a simple text input description and generates a graph. Rather than having the graphing component write to disk, it should also push out an ADIOS stream to some other

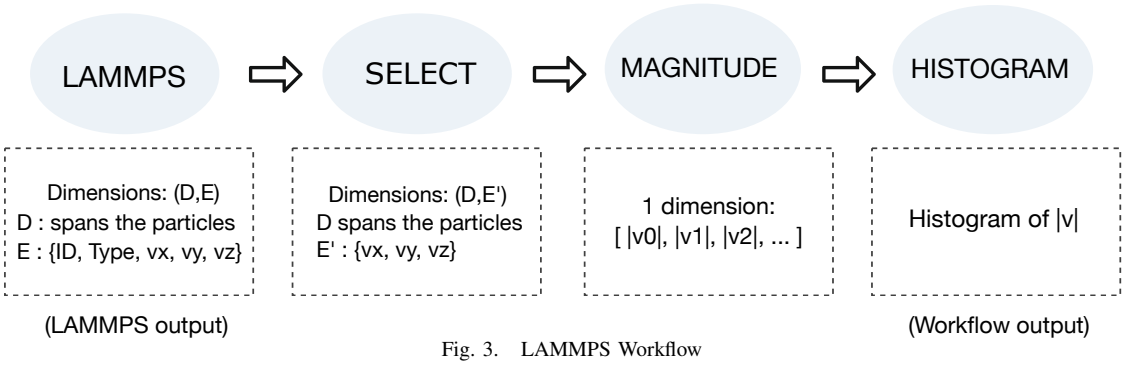

Fig. 3. LAMMPS Workflow

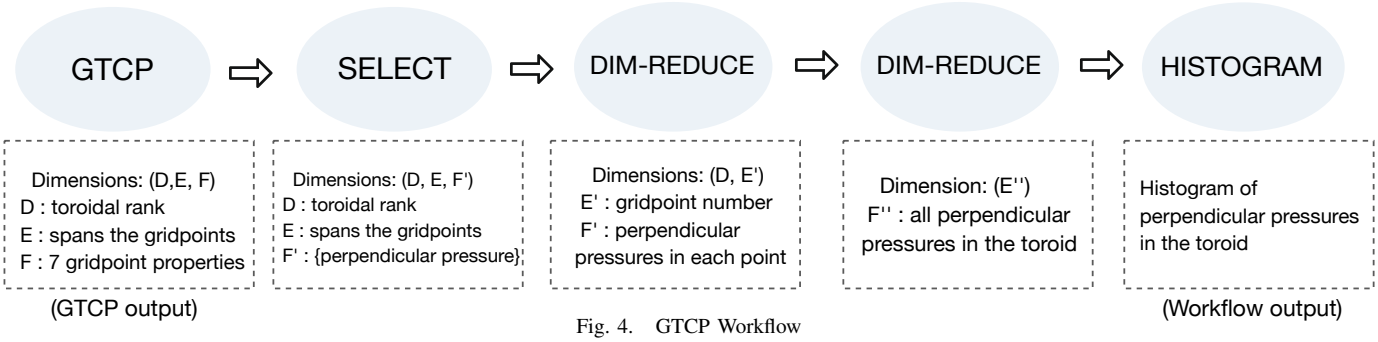

Fig. 4. GTCP Workflow

consumer. An additional Dumper that writes an image file in a particular format, such as JPEG, PNG, or SVG, would be a valuable addition.

VI. Integrating Reusable Components

In order to use some of the same components in both workflows, we had to slightly modify the output stages of the scientific codes driving them. Because in both workflows, the first component to receive the simulation data is Select, each simulation has to write a header of its quantities in the dimension to be selected from. Also, normally LAMMPS packs its two-dimensional output into a single array. We eliminated this packing to reduce the downstream decoding complexity by leaving the actual data structure rather than an encoded form. Both simulations had to be modified to use ADIOS for output. While the ADIOS integration was not difficult because the ADIOS interface is simple, it was the most significant change made to the simulations.

In general, in order to work with SuperGlue components, simulations have to specify through ADIOS the logical dimensions of their data and optionally create corresponding headers to label them and their indices.

A. Demonstrating in the Workflows

We built the LAMMPS workflow, illustrated in Figure 3, using only LAMMPS and SuperGlue components. We annotate the figure with details about how the data is manipulated at each step.

Data arrives from LAMMPS at the first SuperGlue component, Select, which extracts the velocity components from the raw output of the simulation. From Select, data is sent to Magnitude, which computes the magnitudes of the velocities. Magnitude outputs one-dimensional data, an array of the magnitudes it calculates, to the final component, Histogram, which expects one-dimensional data as input. The end result of this workflow is a series of histograms of the total velocities of the particles. There is one histogram created at each timestep at which the simulation would normally dump its data to disk.

The GTCP workflow is illustrated in Figure 4, and it too was assembled only from the simulation and SuperGlue components. Note that the workflows primarily differ in the data formats output by the simulations.

From GTCP, data first arrives at an instance of Select, which extracts one quantity of interest out of the 7 properties that describe each gridpoint. This quantity was arbitrarily chosen as the "perpendicular pressure," or pressure of the plasma perpendicular to the flow in the grid point of interest. Even if it contains only perpendicular pressures, the output of Select is still three-dimensional since this component maintains the original dimensions of its input. Because the Histogram component expects one-dimensional input, we first send the output of Select through two instances of our Dim-Reduce component, each of which eliminates a single dimension of the array without changing its total size. The final component, Histogram, outputs a histogram of the perpendicular pressures of all grid points at each timestep at which the simulation would normally output its data to disk.

VII. Evaluation

The evaluation is performed on Titan, the Cray XK7 machine at Oak Ridge National Laboratory. It consists of 18,688 nodes each with 1 16-core AMD Opteron CPU and 32 GB of RAM. The interconnect is a Gemini network. There is an attached Nvidia Kepler K20X GPU with an additional 6 GB of memory on every node.

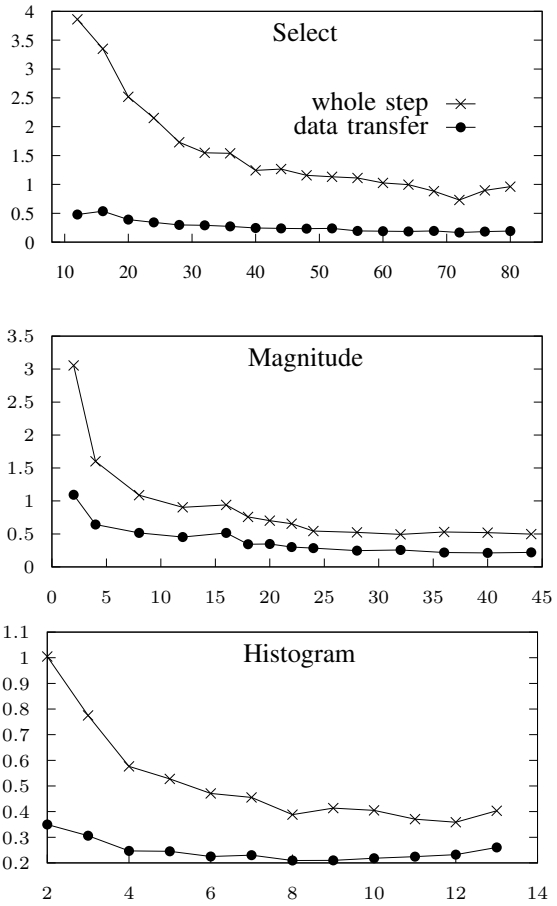

Fig. 5. SuperGlue strong scaling in the LAMMPS workflow. Completion time (secs) of a full timestep and of the data transfer portion of the same timestep are plotted against process size.

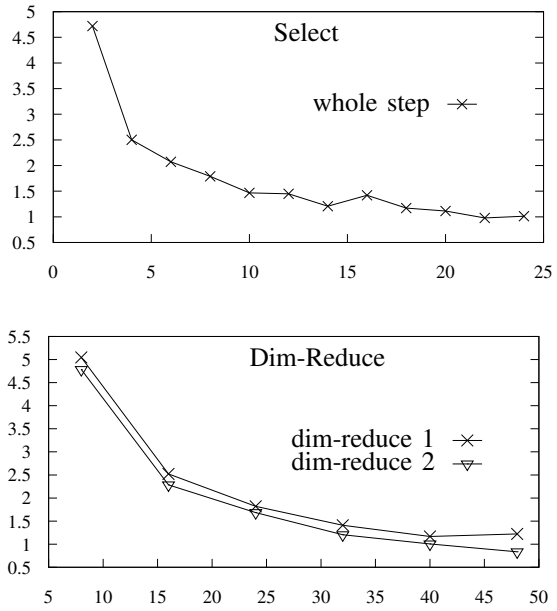

Fig. 6. SuperGlue strong scaling in the GTCP workflow. Whole timestep completion time (secs) for Select and both instances of Dim-Reduce used in the workflow are plotted against process size

A. *Strong Scaling Experiments*

To understand the strong scaling behavior exhibited by the components in different scenarios, we carried out strong scaling measurements of the components in both the LAMMPS and GTCP workflows. To do this, we varied the process size of a single component at a time while fixing that of the other components involved in the workflow, and using a fixed output size from the driving simulation. We determined reasonable process sizes for the fixed-size components using preliminary testing.

The results are illustrated in Figure 5 and Figure 6. Each point shows the completion time for a single time step arbitrarily chosen in the middle of the execution of the workflow. Depicted below the strong scaling curves in Figure 5 are the data transfer times. That is, these points show the portion of the timestep completion time spent by the components waiting to receive requested data. The workflow configurations (process counts) used to obtain these measurements are shown in Table I and Table II. The global output of the simulation in the LAMMPS workflow is 1.28 GB. In the GTCP workflow, the Select measurements are taken using a 900 MB output from the simulation, and those of Dim-Reduce using a 3.8 GB output. The scale used for these results is small compared to the scale at which these simulations can be run due to the limited time and resources available for the numerous complete workflows executions required to obtain meaningful strong scaling results.

These results provide valuable information about the components. First, they show that the components exhibit regular strong scaling behavior. That is, a linear domain of scalability is followed by a turning point and an eventual flattening out of the performance, where the benefit of adding more processes dwindles. That there exists a linear domain means that given a particular workload, users can select SuperGlue process sizes that match their resources and performance requirements. The flat domain is long and does not show any drastic reversal of performance. Without knowing the full strong scaling characteristics of the SuperGlue components used in a particular workflow, a user can safely guess a process size to use for a particular component, using the strong scaling results from the same component with a similar workload size, without incurring unreasonable overhead.

The turning point of scalability of a component is not necessarily determined by its per-process workload size. In Figure 5, the turning point of Select occurs at a per-process workload of around 32 MB. Here, the global data size sent to Select is 1.28 GB. In a separate set of measurements using the GTCP workflow, the turning point of Select occurs at a per-process workload of 113 MB, where the global dataset size was 3.8 GB. Therefore, global workload size is a factor in determining the turning point of scalability of the SuperGlue components.

While there is communication overhead in the computation itself, Figure 5 shows the majority of the communication overhead, i.e., the time spent on data transfer between components, is small compared to the total per-timestep execution time of the SuperGlue components. This supports the idea of assembling workflows using numerous, simple, generic components.

TABLE I
LAMMPS EVALUATION CONFIGURATION SETTINGS

Component Test	LAMMPS Procs	Select Procs	Magnitude Procs	Histogram Procs
Select	256	x	16	8
Magnitude	256	60	x	8
Histogram	256	32	16	x

TABLE II
GTCP EVALUATION CONFIGURATION SETTINGS

Component Test	GTCP Procs	Select Procs	Dim-Reduce 1	Dim-Reduce 2	Histogram Procs
Select	64	x	4	4	4
Dim-Reduce 1	128	32	x	16	16
Dim-Reduce 2	128	32	16	x	16

B. Weak Scaling Experiments

Additional experiments are performed for the GTCP workflow to determine the weak scaling performance for the components. The configurations for these experiments are presented in Table III. The performance results are presented in Table IV. To read the tables, match the rows. For example, the first row in Table III corresponds to the performance results in row 1 in Table IV. These runs use SuperGlue process sizes for which the per-process workloads reside near the turning points in the strong scaling results presented above.

Overall, the components and the overall workflow exhibit very promising weak scaling behavior. While there are slightly different per-process data sizes in each row of the table for both per-simulation process and per-SuperGlue process, there is little variation in the timestep completion times of the SuperGlue components and in the end-to-end completion times of the entire workflow for different workload sizes.

Select gets the full brunt of the total data size. Dividing the process count into the total data size shows that on average, it is roughly the same for weak scaling. The other components exhibit similar performance consistency. Also note that these are not exactly identical ratios or counts. The data size per GTCP process is maintained as it is scaled. The amount of data each SuperGlue component process varies a bit. Also note that there is not a fixed n-1 ratio required for any of the components. Instead, an m-n mapping works correctly.

VIII. CONCLUSIONS AND FUTURE WORK

This paper presents SuperGlue, a demonstration of making generic, reusable components for scientific simulations. By decomposing the operations into small chunks, we can achieve components that can be reused, without modification, for a variety of different workflows. In this work we investigate using a stream-based structure with generic components to achieve easier to build and use workflows. Stream-based, generic workflow components should be designed so as to allow for the greatest variety in their arrangement and for a maximum number of downstream subscribers. Designing components with the ability to handle data having any number of dimensions provides a very useful way to link them together. Maintaining a high level of semantics upstream, for example by labeling dimensions and certain quantities inside of these dimensions, gives a good understanding of the data to downstream components. There is a need for components that organize the data in a format that downstream components can understand. And, designing specific disk writer components removes the need to temporarily modify analytics components to let them also act as disk writers.

Through the demonstration of generating a velocity histogram for LAMMPS, the molecular dynamics simulation, and a pressure histogram for GTC, the particle in cell fusion reactor simulator, we achieved reusing the same components in very different data formats and application types.

While this work leverages ADIOS and the FlexPath transport, this is not the only approach for addressing reusable components. Other, similar approaches can also work well. However, in this case, the data annotation provided by this connection infrastructure helps enable reusable components by offering the necessary metadata to perform the general operations.

There are a number of improvements we can make to our current implementation to have at our disposal more robust and flexible workflow components. As mentioned earlier, reading and writing dimension labels at each step in the workflow provides more information to downstream components and presents a clear advantage. The ADIOS interface includes the ability to send output to multiple destinations by having several "write groups." We wish to explore the possibility of a *Fork Component* that would use this functionality of ADIOS to allow the creation of branched workflows.

The components we have developed in this research cover only a small portion of the procedures that computational scientists need for their workflows. Eventually, we wish to allow for the development of a large collection of generic workflow components. We can take steps in this direction by building on our existing components. For example, *Magnitude* performs a relatively simple operation on multi-dimensional data, where one dimension spans a number of quantities involved in each instance of the operation. This model can fit any number of operations involving a repeating, fixed number of quantities, and it can even be made to work with a formula specified by the user at runtime. This opens the door to a large family of generic components.

Finally, while we have kept performance in mind in the development of these components, performance optimization is not yet the focus of this research. In the design of any generic tool however, the question of performance inevitably arises. Indeed, designing tools that are not meant to operate

TABLE III
GTCP WEAK SCALING EVALUATION CONFIGURATION SETTINGS

GTCP Procs	Select Procs	Dim-Reduce 1	Dim-Reduce 2	Histogram Procs	Total Data Size	End-to-End Time
64	10	6	6	2	918,303,680	92.724
84	16	10	10	2	1,434,599,936	115.232
156	18	14	14	4	2,065,583,520	97.266
234	25	19	19	5	2,811,256,000	96.359

TABLE IV
GTCP WEAK SCALING COMPONENT PERFORMANCE

Select Average Time	Dim-Reduce 1 Average Time	Dim-Reduce 2 Average Time	Histogram Average Time
1.55	1.58	1.34	0.6
2.34	1.58	1.72	0.78
2.31	1.73	1.54	0.713
2.19	1.76	1.68	0.89

on a specific format of input data can easily impact performance. For example, *Dim-Reduce* performs the same amount of computation whether it re-arranges data or not. In the long run, optimizing these components will involve detecting such situations where they can avoid performing unnecessary iterations and data manipulation.

ACKNOWLEDGMENTS

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

This work was supported by Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357, program manager Lucy Nowell.

REFERENCES

[1] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: scalable data staging services for petascale applications. In D. Kranzlmüller, A. Bode, H.-G. Hegering, H. Casanova, and M. Gerndt, editors, *HPDC*, pages 39–48. ACM, 2009.

[2] M. Chaarawi. HDF5 Virtual Object Layer, 2013. http://tinyurl.com/qj3glq5.

[3] J. Dayal, D. Bratcher, H. Abbasi, G. Eisenhauer, S. Klasky, N. Podhorszki, K. Schwan, and M. Wolf. Flexpath: Type-Based Publish/Subscribe System for Large-scale Science Analytics. In *Cluster, Cloud, and Grid*, CCGrid '14. IEEE, 2014.

[4] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. Podhorszki. Flexpath: Type-based publish/subscribe system for large-scale science analytics. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 246–255. IEEE, 2014.

[5] M. Dreher and T. Peterka. Bredala: Semantic data redistribution for in situ applications. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Ieee, 2016.

[6] G. Eisenhauer, M. Wolf, H. Abbasi, S. Klasky, and K. Schwan. A type system for high performance communication and computation. In *e-Science Workshops (eScienceW), 2011 IEEE Seventh International Conference on*, pages 183–190. IEEE, 2011.

[7] H. Karimabadi, B. Loring, P. O'Leary, A. Majumdar, M. Tatineni, and B. Geveci. In-situ visualization for global hybrid simulations. In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, page 57. ACM, 2013.

[8] Z. Lin, T. S. Hahm, W. W. Lee, W. M. Tang, and R. B. White. Turbulent transport reduction by zonal flows: Massively parallel simulations. *Science*, 281(5384):1835–1837, September 1998.

[9] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich IO methods for portable high performance IO. In *Proceedings of the International Parallel and Distributed Processing Symposium*, Rome, Italy, 2009.

[10] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.

[11] G. Malewicz, I. Foster, A. Rosenberg, and M. Wilde. A tool for prioritizing DAGMan jobs and its evaluation. *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pages 156–168, 0-0 2006.

[12] McKesson. Formfast, 2016. http://formfast.com/platform/integrate/ehr-integration/mckesson/.

[13] K. Moreland, D. Lepage, D. Koller, and G. Humphreys. Remote rendering for ultrascale data. *Journal of Physics: Conference Series*, 125(1):012096, 2008.

[14] S. J. Mullender, I. M. Leslie, and D. McAuley. Operating-system support for distributed multimedia. In *Proceedings of the 1994 Summer USENIX Technical Conference*, pages 209–219, 1994.

[15] A. Nisar, W.-k. Liao, and A. Choudhary. Scaling parallel I/O performance through I/O delegate and caching system. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[16] C. C. Ober, R. A. Oldfield, D. E. Womble, and J. V. Dyke. Seismic imaging on the Intel Paragon. *Computers & Mathematics with Applications*, 35(7):65 – 72, 1998. Advanced Computing on Intel Architectures.

[17] R. A. Oldfield, P. Widener, A. B. Maccabe, L. Ward, and T. Kordenbrock. Efficient data-movement for lightweight I/O. In *Proceedings of the 2006 International Workshop on High Performance I/O Techniques and Deployment of Very Large Scale I/O Systems*, Barcelona, Spain, Sept. 2006.

[18] S. Plimpton, R. Pollock, and M. Stevens. Particle-mesh ewald and rrespa for parallel molecular dynamics simulations. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, PPSC 1997, March 14-17, 1997, Hyatt Regency Minneapolis on Nicollet Mall Hotel, Minneapolis, Minnesota, USA*. SIAM, 1997.

[19] J. Racine. gnuplot 4.0: a portable interactive plotting utility. *Journal of Applied Econometrics*, 21(1):133–141, 2006.

[20] M. Riedel, T. Eickermann, S. Habbinga, W. Frings, P. Gibbon, D. Mallmann, F. Wolf, A. Streit, T. Lippert, W. Schiffmann, A. Ernst, R. Spurzem, and W. Nagel. Computational steering and online visualization of scientific applications on large-scale hpc systems within e-science infrastructures. In *e-Science and Grid Computing, IEEE International Conference on*, pages 483 –490, dec. 2007.

[21] J. Soumagne, D. Kimpe, J. A. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, and R. B. Ross. Mercury: Enabling remote procedure call for high-performance computing. In *CLUSTER*, pages 1–8. IEEE, 2013.

[22] V. Vishwanath, M. Hereld, and M. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 9 –14, oct. 2011.

[23] V. Vishwanath, M. Hereld, and M. E. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 9–14. IEEE, 2011.

[24] wfmc. Workflow management coalition, 2016. http://wwww.wfmc.org/.

[25] B. Whitlock, J. M. Favre, and J. S. Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, EGPGV '11, pages 101–109, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.

[26] Yassine Mrabet. Simple torus. https://commons.wikimedia.org/wiki/File:Simple_Torus.svg, 2007. Accessed: April 27, 2016.

[27] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA - preparatory data analytics on Peta-Scale machines. In *In Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium, April, Atlanta, Georgia*, 2010.